ETC1010: Introduction to Data Analysis

Week of Tidy Data + Style

Stuart Lee & Nicholas Tierney

11th Mar 2020

How to learn

I want to some time to discuss ideas on learning, and how it ties into the course.

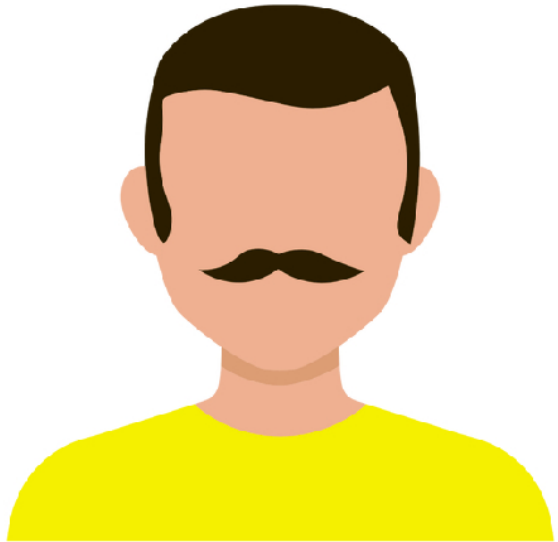
Beginner

"I don't know what I don't know."

Competent Practitioner

"I can do it, but I may look things up."

Expert



"I can handle anything you throw at me"

Beginner

"I don't know what I don't know."

Competent Practitioner

"I can do it, but I may look things up."

Expert

"I can handle anything you throw at me"

Beginner

Competent Practitioner

Expert

"I don't know what I don't know."

"I can do it, but I may look things up."

"I can handle anything you throw at me"

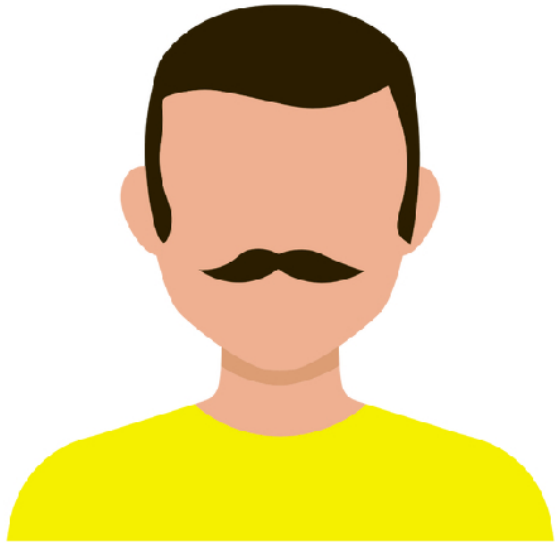
Beginner

"I don't know what I don't know."

Competent Practitioner

"I can do it, but I may look things up."

Expert



"I can handle anything you throw at me"

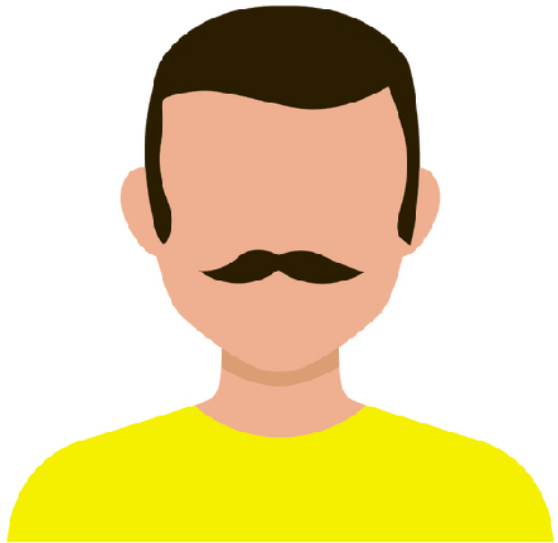
Beginner

No mental model

Competent Practitioner

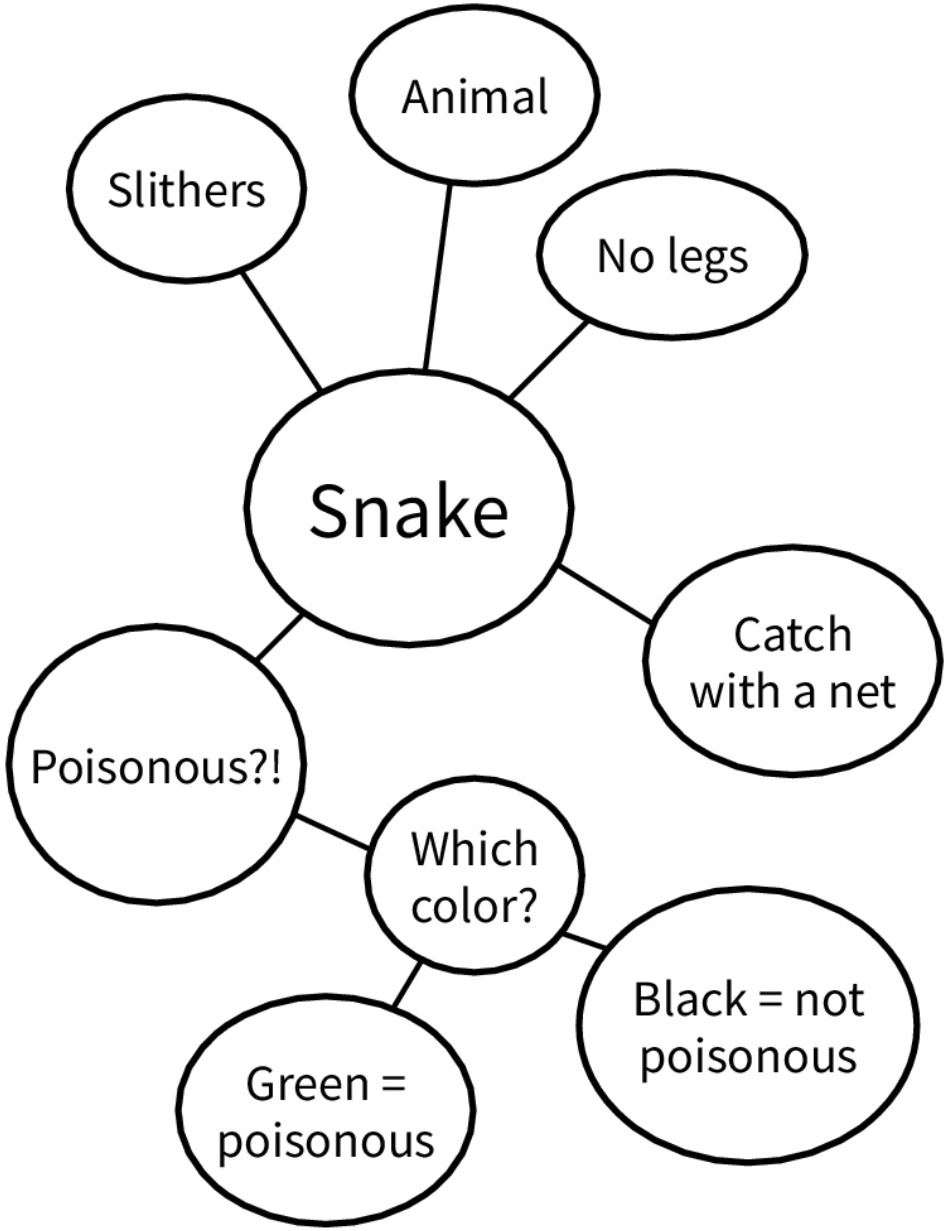
Useful mental model

Expert



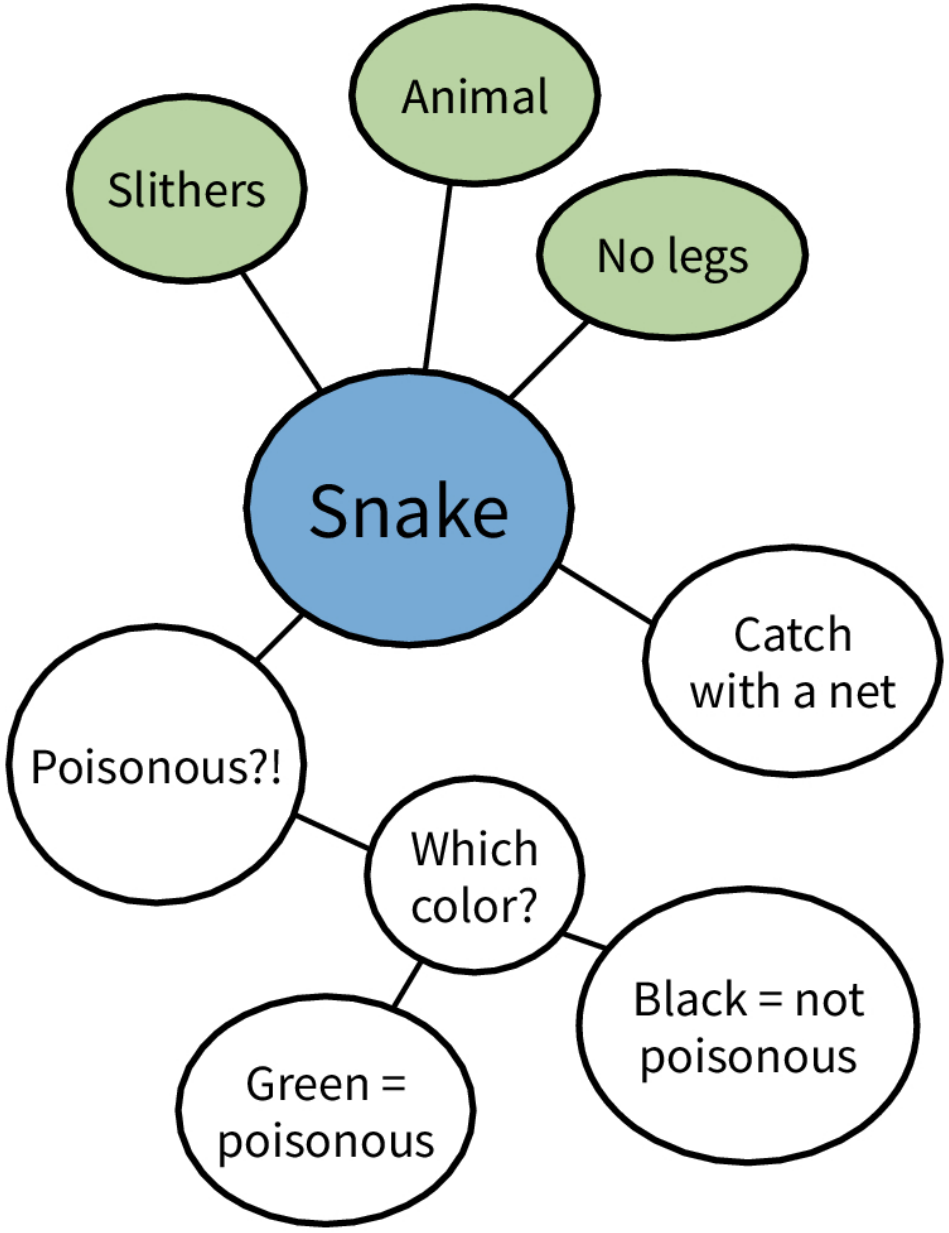
Elaborate mental models

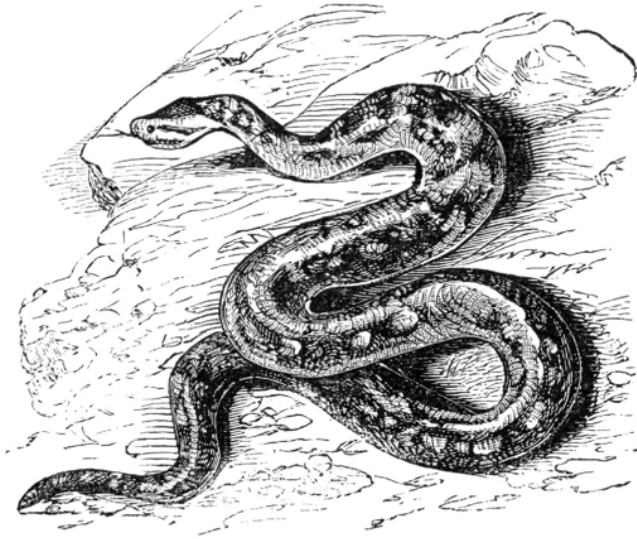
Mental Models

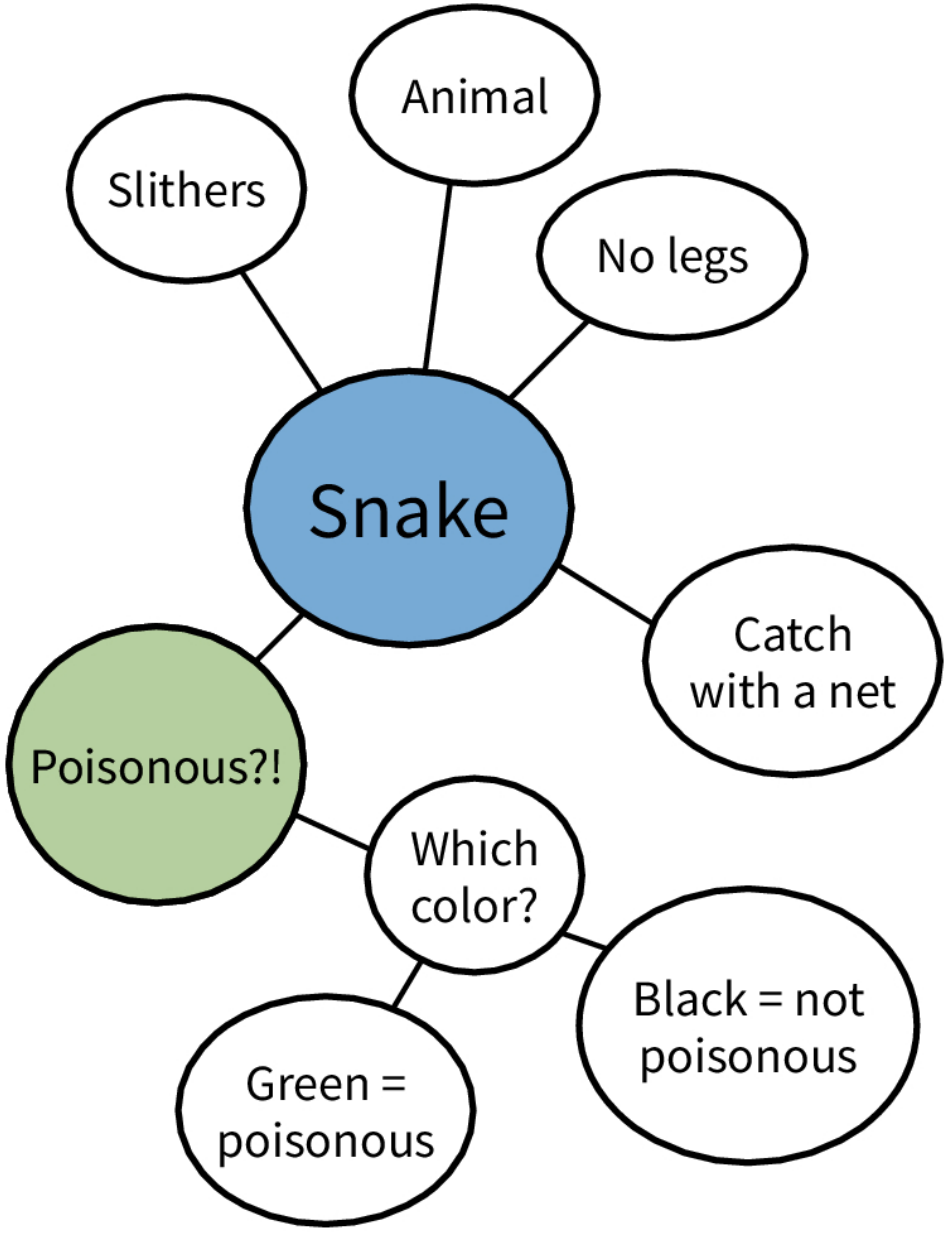


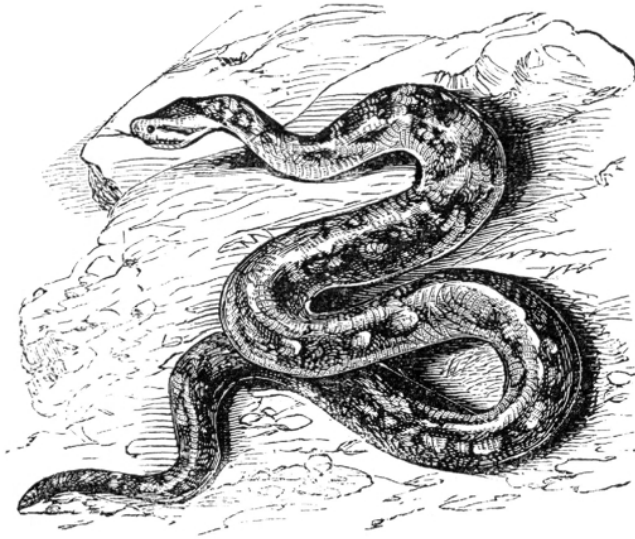
Mental Model

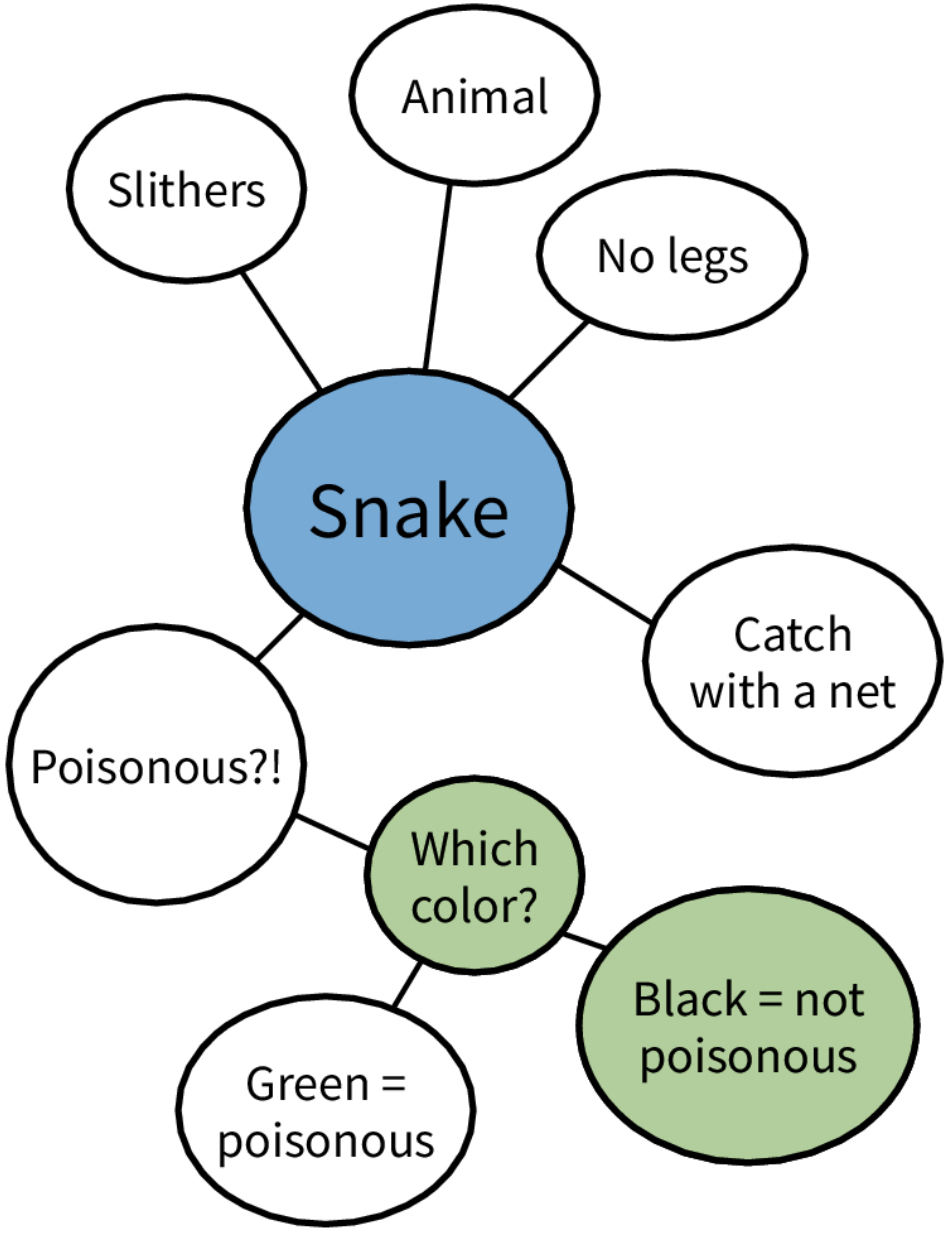
a structure that organizes facts according to their relationships

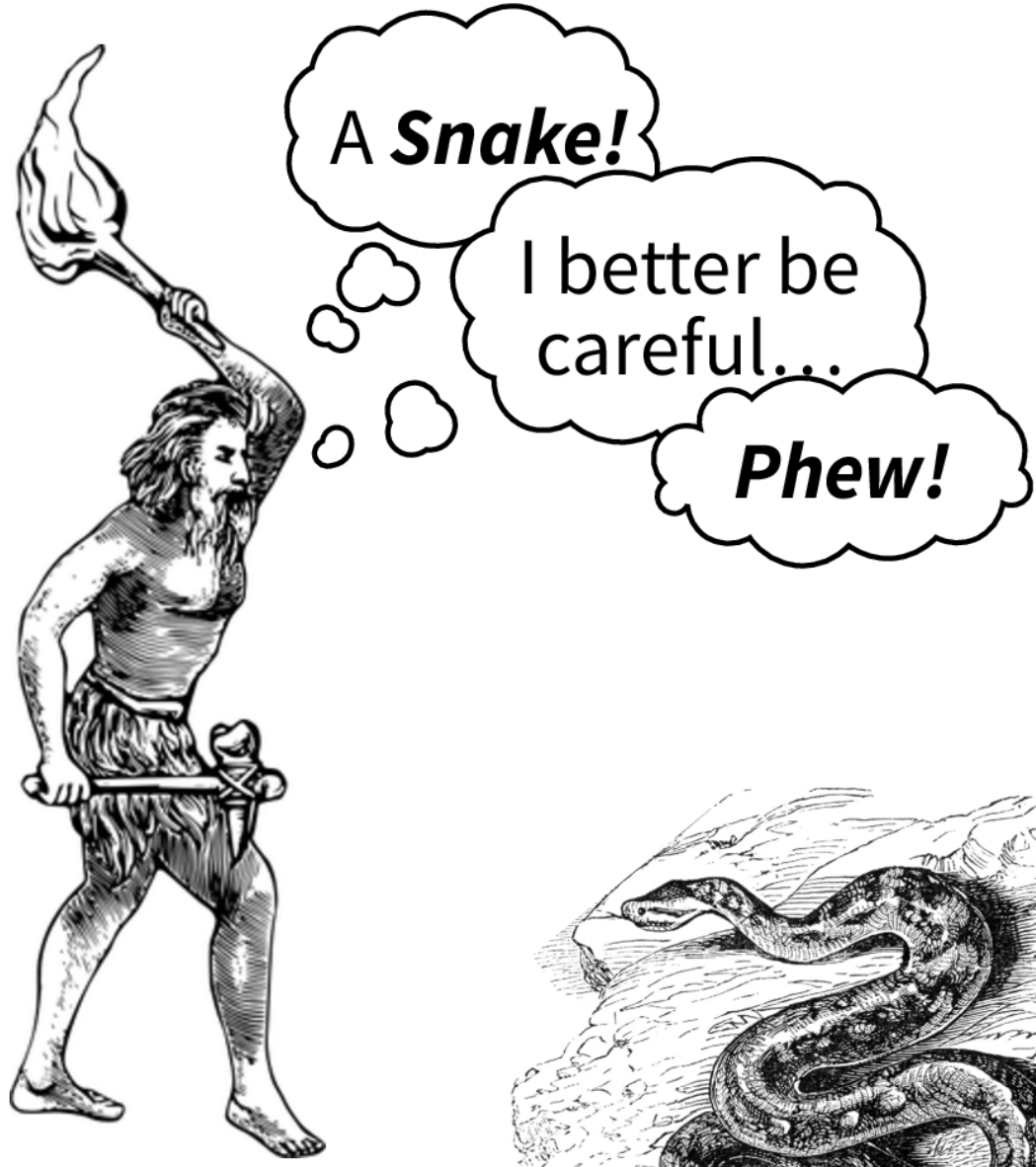


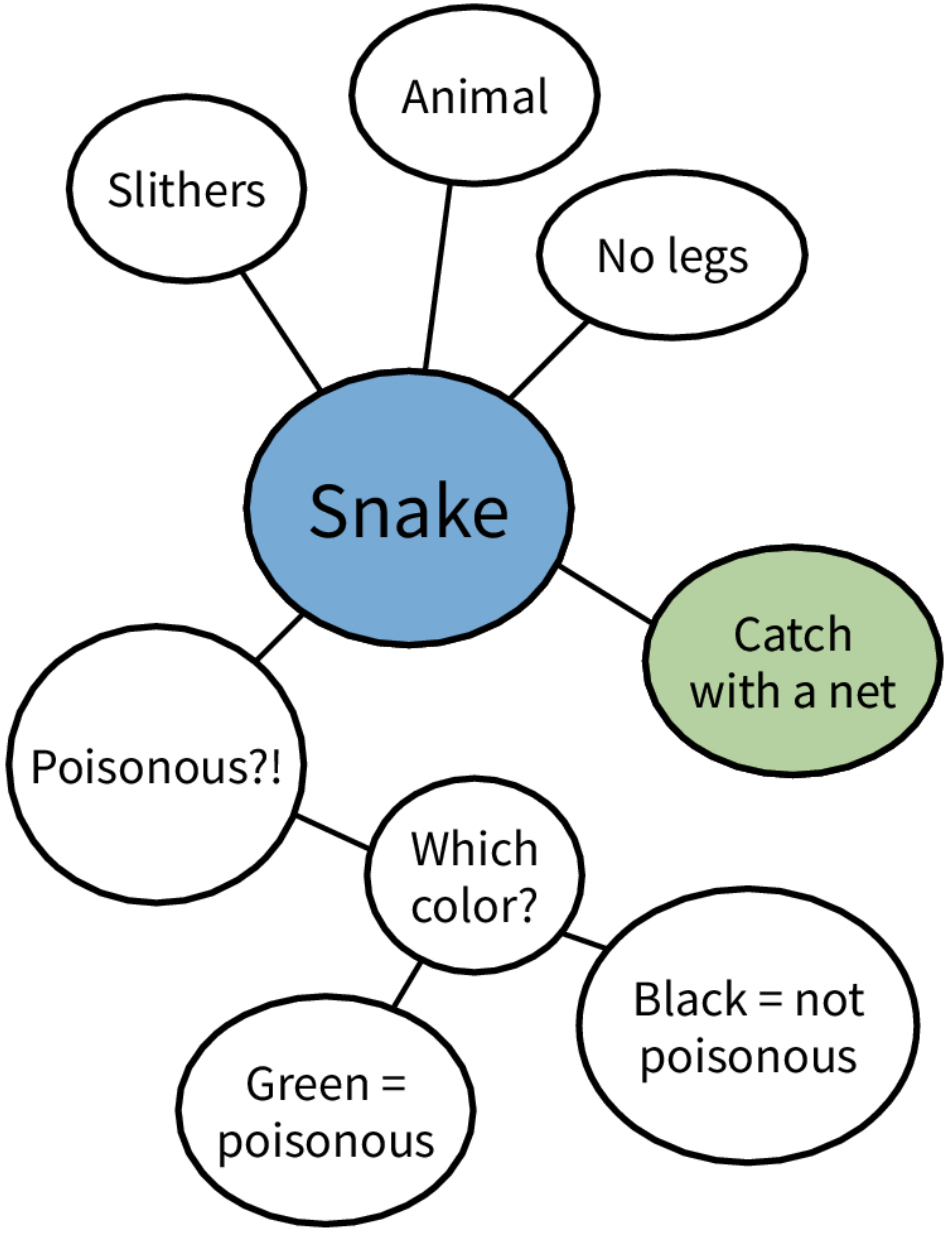


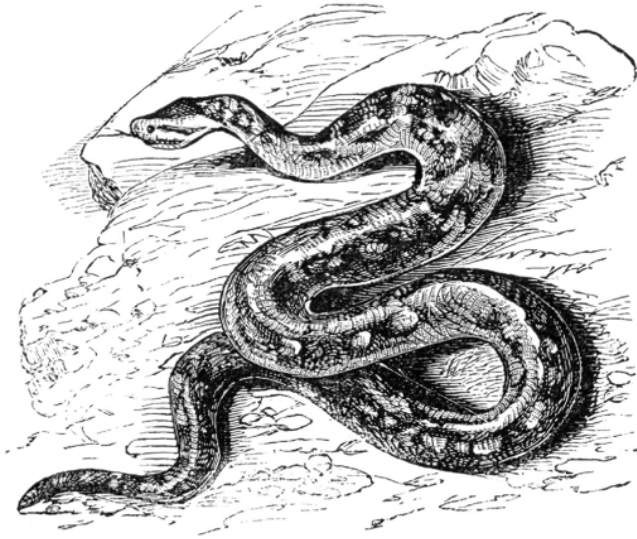












Mental Model

a structure that organizes facts according to their relationships

(demo)

recap

- R + Rstudio
- Functions are _
- columns in data frames are accessed with _ ?
- packages are installed with _ ?
- packages are loaded with _ ?

- Why do we care about Reproducibility?
- Output + input of rmarkdown
- I have an assignment group
- I have made contact with my assignment group

Style guide

> *"Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread." -- Hadley Wickham*

- Style guide for this course is based on the Tidyverse style guide: http://style.tidyverse.org/
- There's more to it than what we'll cover today, we'll mention more as we introduce more functionality, and do a recap later in the semester

File names and code chunk labels

- Do not use spaces in file names, use - or _ to separate words
- Use all lowercase letters

```
# Good
ucb-admit.csv

# Bad
UCB Admit.csv
```

Object names

- Use _ to separate words in object names
- Use informative but short object names
- Do not reuse object names within an analysis

```
# Good
acs_employed

# Bad
acs.employed
acs2
acs_subset
acs_subsetted_for_males
```

Spacing

- Put a space before and after all infix operators (=, +, -, <-, etc.), and when naming arguments in function calls.
- Always put a space after a comma, and never before (just like in regular English).

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)

# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
```

ggplot

- Always end a line with +
- Always indent the next line

```
# Good
ggplot(diamonds, mapping = aes(x = price)) +
  geom_histogram()

# Bad
ggplot(diamonds,mapping=aes(x=price))+geom_histog
```

Long lines

- Limit your code to 80 characters per line. This fits comfortably on a printed page with a reasonably sized font.
- Take advantage of RStudio editor's auto formatting for indentation at line breaks.

Assignment

- Use <- not =

```r
# Good
x <- 2

# Bad
x = 2
```

Quotes

Use ", not ', for quoting text. The only exception is when the text already contains double quotes and no single quotes.

```
ggplot(diamonds, mapping = aes(x = price)) +
  geom_histogram() +
  # Good
  labs(title = "`Shine bright like a diamond`",
  # Good
       x = "Diamond prices",
  # Bad
       y = 'Frequency')
```

Source: Artwork by @allison_horst

Overview

- `filter()`
- `select()`
- `mutate()`
- `arrange()`
- `group_by()`
- `summarise()`
- `count()`

Artwork by @allison_horst

R Packages

```
avail_pkg <- available.packages(contriburl = cont
dim(avail_pkg)
## [1] 15383    17
```

As of 2020-03-17 there are 15383 R packages available

Name clashes

```
library(tidyverse)
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
## ✓ ggplot2 3.3.0.9000     ✓ purrr   0.3.3
## ✓ tibble  2.1.3          ✓ dplyr   0.8.5
## ✓ tidyr   1.0.2          ✓ stringr 1.4.0
## ✓ readr   1.3.1          ✓ forcats 0.4.0
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter()     masks stats::filter()
## x dplyr::group_rows() masks kableExtra::group_r
## x dplyr::lag()        masks stats::lag()
```

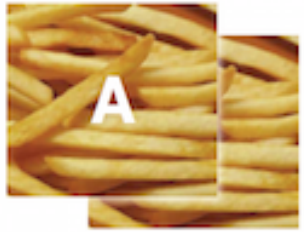
Many R packages

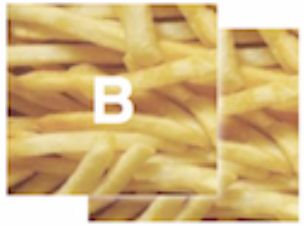
- A blessing & a curse!
- So many packages available, it can make it hard to choose!
- Many of the packages are designed to solve a specific problem
- The tidyverse is designed to work with many other packages following a consistent philosophy
- What this means is that you shouldn't notice it!

Let's talk about data

12 subjects

Three oils,
two batches

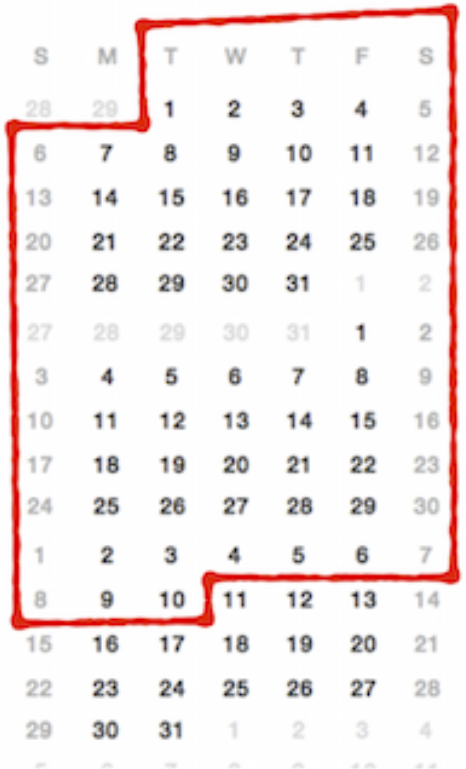




Five scales

RANCID

For 10 weeks



Example: french fries

- Experiment in Food Sciences at Iowa State University.
- Aim: find if cheaper oil could be used to make hot chips
- Question: Can people distinguish between chips fried in the new oils relative to those current market leader oil.
- 12 tasters recruited
- Each sampled two chips from each batch
- Over a period of ten weeks.

Same oil kept for a period of 10 weeks! May be a bit gross!

Example: french-fries - pivoting into long form

```
french_fries <- read_csv("data/french_fries.csv")
french_fries
```

```
## # A tibble: 6 x 9
##    time treatment subject   rep potato buttery
##   <dbl>     <dbl>   <dbl> <dbl>  <dbl>   <dbl>
## 1     1         1       3     1    2.9       0
## 2     1         1       3     2   14         0
## 3     1         1      10     1   11         6.4
## 4     1         1      10     2    9.9       5.9
## 5     1         1      15     1    1.2       0.1
## 6     1         1      15     2    8.8       3
```

Example: french-fries - pivoting into long form

```
fries_long <- french_fries %>%
  pivot_longer(cols = potato:painty,
               names_to = "type",
               values_to = "rating") %>%
  mutate(type = as.factor(type))
fries_long
## # A tibble: 3,480 x 6
##    time treatment subject   rep type    rating
##   <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
## 1     1         1       3     1 potato     2.9
## 2     1         1       3     1 buttery    0
## 3     1         1       3     1 grassy     0
```

`filter()`

choose observations from your data

filter(): example

```
fries_long %>%
  filter(subject == 10)
```

```
## # A tibble: 300 x 6
##    time treatment subject   rep type    rating
##   <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
## 1     1         1      10     1 potato    11
## 2     1         1      10     1 buttery    6.4
## 3     1         1      10     1 grassy     0
## 4     1         1      10     1 rancid     0
## 5     1         1      10     1 painty     0
## 6     1         1      10     2 potato     9.9
## 7     1         1      10     2 buttery    5.9
```

`filter()`: details

Filtering requires comparison to find the subset of observations of interest. What do you think the following mean?

- `subject != 10`
- `x > 10`
- `x >= 10`
- `class %in% c("A", "B")`
- `!is.na(y)`

03:00

`filter()`: details

```
subject != 10
```

Find rows corresponding to all subjects except subject 10

```
x > 10
```

find all rows where variable x has values bigger than 10

```
x >= 10
```

finds all rows variable x is greater than or equal to 10.

```
class %in% c("A", "B")
```

finds all rows where variable `class` is either A or B

```
!is.na(y)
```

finds all rows that *DO NOT* have a missing value for variable y

Your turn: open french-fries.Rmd

Filter the french fries data to have:

- only week 1
- oil type 1 (oil type is called treatment)
- oil types 1 and 3 but not 2
- weeks 1-4 only

French Fries Filter: only week 1

```
fries_long %>% filter(time == 1)
## # A tibble: 360 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
```

French Fries Filter: oil type 1

```
fries_long %>% filter(treatment == 1)
## # A tibble: 1,160 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
```

French Fries Filter: oil types 1 and 3 but not 2

```
fries_long %>% filter(treatment != 2)
```

```
## # A tibble: 2,320 x 6
##     time treatment subject   rep type    rating
##    <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
##  1     1         1       3     1 potato     2.9
##  2     1         1       3     1 buttery    0
##  3     1         1       3     1 grassy     0
##  4     1         1       3     1 rancid     0
##  5     1         1       3     1 painty     5.5
##  6     1         1       3     2 potato    14
##  7     1         1       3     2 buttery    0
##  8     1         1       3     2 grassy     0
```

French Fries Filter: weeks 1-4 only

```
fries_long %>% filter(time %in% c("1", "2", "3",
## # A tibble: 1,440 x 6
##    time treatment subject   rep type    rating
##   <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
## 1     1         1       3     1 potato     2.9
## 2     1         1       3     1 buttery    0
## 3     1         1       3     1 grassy     0
## 4     1         1       3     1 rancid     0
## 5     1         1       3     1 painty     5.5
## 6     1         1       3     2 potato    14
## 7     1         1       3     2 buttery    0
## 8     1         1       3     2 grassy     0
```

about `%in%`

[demo]

`select()`

- Chooses which variables to keep in the data set.
- Useful when there are many variables but you only need some of them for an analysis.

`select()`: a comma separated list of variables, by name.

```
french_fries %>%
  select(time,
         treatment,
         subject)
## # A tibble: 696 x 3
##     time treatment subject
##    <dbl>     <dbl>   <dbl>
##  1     1         1       3
##  2     1         1       3
##  3     1         1      10
##  4     1         1      10
##  5     1         1      15
```

`select()`: drop selected variables by prefixing with `-`

```
french_fries %>%
  select(-time,
         -treatment,
         -subject)
```

```
## # A tibble: 696 x 6
##      rep potato buttery grassy rancid painty
##    <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
##  1     1    2.9       0      0      0    5.5
##  2     2   14         0      0      1.1  0
##  3     1   11         6.4    0      0    0
##  4     2    9.9       5.9    2.9    2.2  0
##  5     1    1.2       0.1    0      1.1  5.1
```

select()

Inside `select()` you can use text-matching of the names like `starts_with()`, `ends_with()`, `contains()`, `matches()`, or `everything()`

```
french_fries %>%
  select(contains("e"))
## # A tibble: 696 x 5
##     time treatment subject   rep buttery
##    <dbl>     <dbl>   <dbl> <dbl>   <dbl>
##  1     1         1       3     1     0
##  2     1         1       3     2     0
##  3     1         1      10     1     6.4
```

select(): Using it

You can use the colon, `:`, to choose variables in order of the columns

```
french_fries %>%
  select(time:subject)
## # A tibble: 696 x 3
##     time treatment subject
##    <dbl>     <dbl>   <dbl>
##  1     1         1       3
##  2     1         1       3
##  3     1         1      10
##  4     1         1      10
##  5     1         1      15
```

Your turn: back to the french fries data

- `select()` time, treatment and rep
- `select()` subject through to rating
- drop subject

03:00

Artwork by @allison_horst

mutate(): create a new variable; keep existing ones

```
french_fries
```

```
## # A tibble: 696 x 9
##     time treatment subject   rep potato buttery
##    <dbl>     <dbl>   <dbl> <dbl>  <dbl>   <dbl>
##  1     1         1       3     1    2.9     0
##  2     1         1       3     2   14       0
##  3     1         1      10     1   11       6.4
##  4     1         1      10     2    9.9     5.9
##  5     1         1      15     1    1.2     0.1
##  6     1         1      15     2    8.8     3
##  7     1         1      16     1    9       2.6
##  8     1         1      16     2    8.2     4.4
```

mutate(): create a new variable; keep existing ones

```
french_fries %>%
  mutate(rainty = rancid + painty)
```

```
## # A tibble: 696 x 10
##     time treatment subject   rep potato buttery
##    <dbl>     <dbl>   <dbl> <dbl>  <dbl>   <dbl>
##  1     1         1       3     1    2.9     0
##  2     1         1       3     2   14       0
##  3     1         1      10     1   11       6.4
##  4     1         1      10     2    9.9     5.9
##  5     1         1      15     1    1.2     0.1
##  6     1         1      15     2    8.8     3
##  7     1         1      16     1    9       2.6
```

Your turn: french fries

Compute a new variable called `lrating` by taking a log of the rating

02:00

`summarise()`: boil data down to one row observation

```
fries_long
```

```
## # A tibble: 6 x 6
##    time treatment subject   rep type    rating
##   <dbl>     <dbl>   <dbl> <dbl> <fct>    <dbl>
## 1     1         1       3     1 potato     2.9
## 2     1         1       3     1 buttery    0
## 3     1         1       3     1 grassy     0
## 4     1         1       3     1 rancid     0
## 5     1         1       3     1 painty     5.5
## 6     1         1       3     2 potato    14
```

`summarise()`: boil data down to one row observation

```
fries_long %>%
  summarise(rating = mean(rating, na.rm = TRUE))
## # A tibble: 1 x 1
##   rating
##    <dbl>
## 1   3.16
```

What if we want a summary for each `type`?

use `group_by()`

Using `summarise()` + `group_by()`

Produce summaries for every group:

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 5 x 2
##   type    rating
##   <fct>    <dbl>
## 1 buttery  1.82
## 2 grassy   0.664
## 3 painty   2.52
## 4 potato   6.95
## 5 rancid   3.85
```

Your turn: Back to french-fries.Rmd

- Compute the average rating by subject
- Compute the average rancid rating per week

03:00

french fries answers

```
fries_long %>%
  group_by(subject) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 12 x 2
##    subject rating
##      <dbl>  <dbl>
##  1       3   2.46
##  2      10   4.24
##  3      15   2.16
##  4      16   3.00
##  5      19   4.54
##  6      31   4.00
```

french fries answers

```
fries_long %>%
  filter(type == "rancid") %>%
  group_by(time) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
```

```
## # A tibble: 10 x 2
##     time rating
##    <dbl>  <dbl>
##  1     1   2.36
##  2     2   2.85
##  3     3   3.72
##  4     4   3.60
##  5     5   3.53
```

`arrange()`: orders data by a given variable.

Useful for display of results (but there are other uses!)

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE))
## # A tibble: 5 x 2
##   type    rating
##   <fct>    <dbl>
## 1 buttery  1.82
## 2 grassy   0.664
## 3 painty   2.52
## 4 potato   6.95
## 5 rancid   3.85
```

arrange()

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE)) %>
  arrange(rating)
```

```
## # A tibble: 5 x 2
##   type    rating
##   <fct>    <dbl>
## 1 grassy   0.664
## 2 buttery  1.82
## 3 painty   2.52
## 4 rancid   3.85
## 5 potato   6.95
```

Your turn: french-fries.Rmd - arrange

- Arrange the average rating by type in decreasing order
- Arrange the average subject rating in order lowest to highest.

02:00

arrange() answers

```
fries_long %>%
  group_by(type) %>%
  summarise(rating = mean(rating, na.rm=TRUE)) %>
  arrange(desc(rating))
## # A tibble: 5 x 2
##   type    rating
##   <fct>    <dbl>
## 1 potato   6.95
## 2 rancid   3.85
## 3 painty   2.52
## 4 buttery  1.82
## 5 grassy   0.664
```

arrange() answers

```
fries_long %>%
  group_by(subject) %>%
  summarise(rating = mean(rating, na.rm=TRUE)) %>
  arrange(rating)
```

```
## # A tibble: 12 x 2
##    subject rating
##      <dbl>  <dbl>
##  1      78   1.94
##  2      79   1.94
##  3      15   2.16
##  4       3   2.46
##  5      52   2.72
```

`count()` the number of things in a given column

```
fries_long %>%
  count(type, sort = TRUE)
## # A tibble: 5 x 2
##   type         n
##   <fct>    <int>
## 1 buttery    696
## 2 grassy     696
## 3 painty     696
## 4 potato     696
## 5 rancid     696
```

Your turn: `count()`

- count the number of subjects
- count the number of types

02:00

French Fries: Putting it together to problem solve

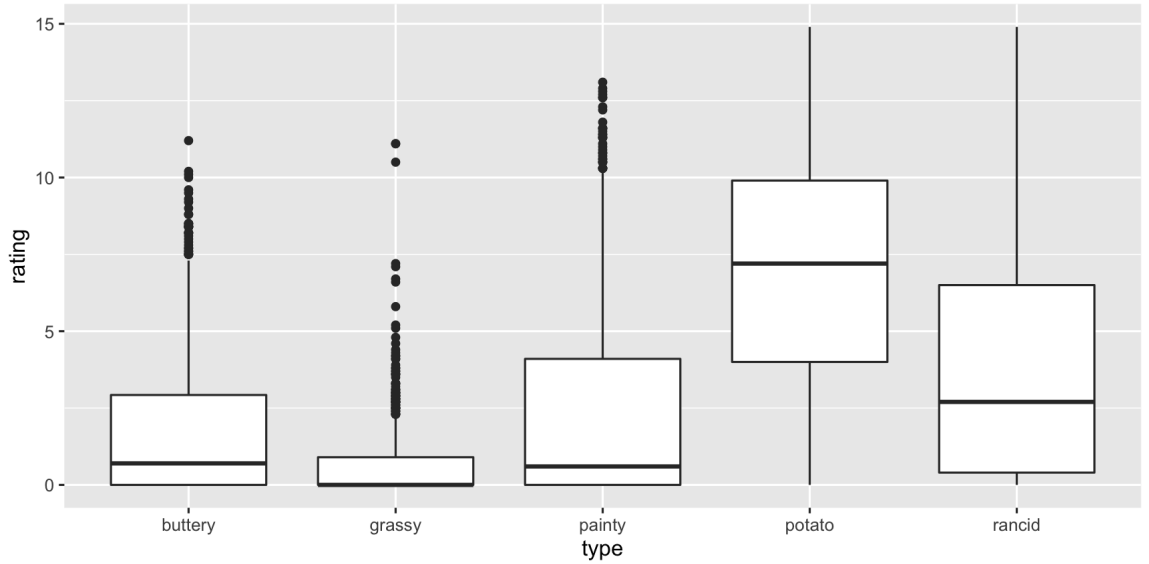
French Fries: Are ratings similar?

```
fries_long %>%
  group_by(type) %>%
  summarise(
    m = mean(rating,
             na.rm = TR
    sd = sd(rating,
            na.rm = TRU
  arrange(-m)
## # A tibble: 5 x 3
##   type          m    s
##   <fct>     <dbl> <dbl
## 1 potato  6.95   3.5
```

The scales of the ratings are quite different. Mostly the chips are rated highly on potato'y, but low on grassy.

French Fries: Are ratings similar?

```
ggplot(fries_long,
       aes(x = type,
           y = rating)) +
  geom_boxplot()
```



French Fries: Are reps like each other?

```
fries_spread <- fries_long %>%
  pivot_wider(names_from = rep,
              values_from = rating)

fries_spread
## # A tibble: 1,740 x 6
##    time treatment subject type      `1`   `2`
##   <dbl>     <dbl>   <dbl> <fct>   <dbl> <dbl>
## 1     1         1       3 potato    2.9  14
## 2     1         1       3 buttery   0     0
## 3     1         1       3 grassy    0     0
## 4     1         1       3 rancid    0     1.1
```

French Fries: Are reps like each other?

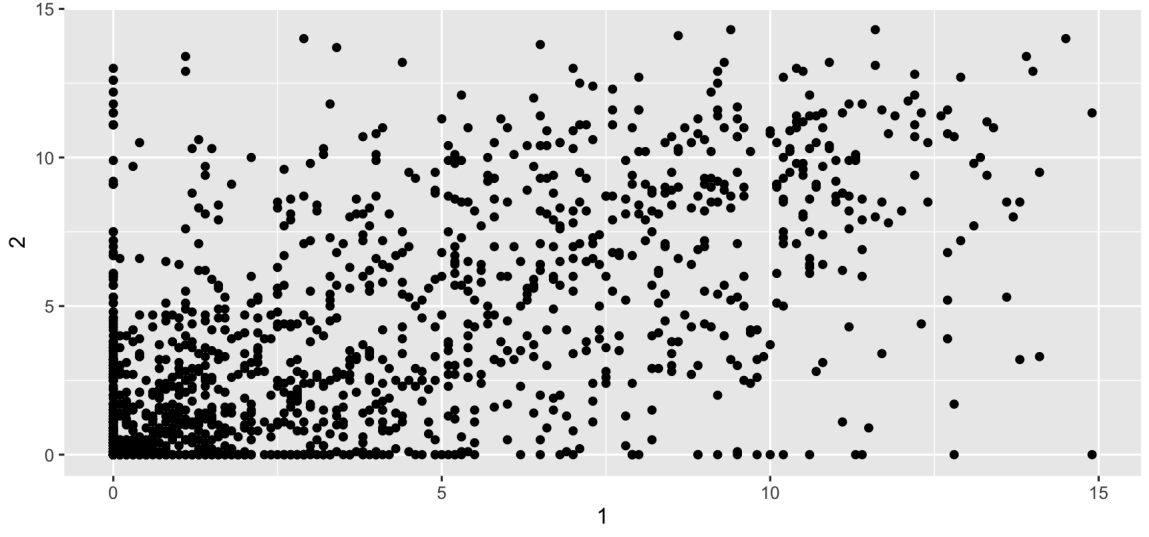
```
summarise(fries_spread,
          r = cor(`1`, `2`, use = "complete.obs")
## # A tibble: 1 x 1
##       r
##   <dbl>
## 1 0.668
```

French Fries:

```
ggplot(fries_spread,
       aes(x = `1`,
           y = `2`)) +
geom_point() +
labs(title = "Data is poor quality: the replica
```

French Fries:

Data is poor quality: the replicates do not look like each other!



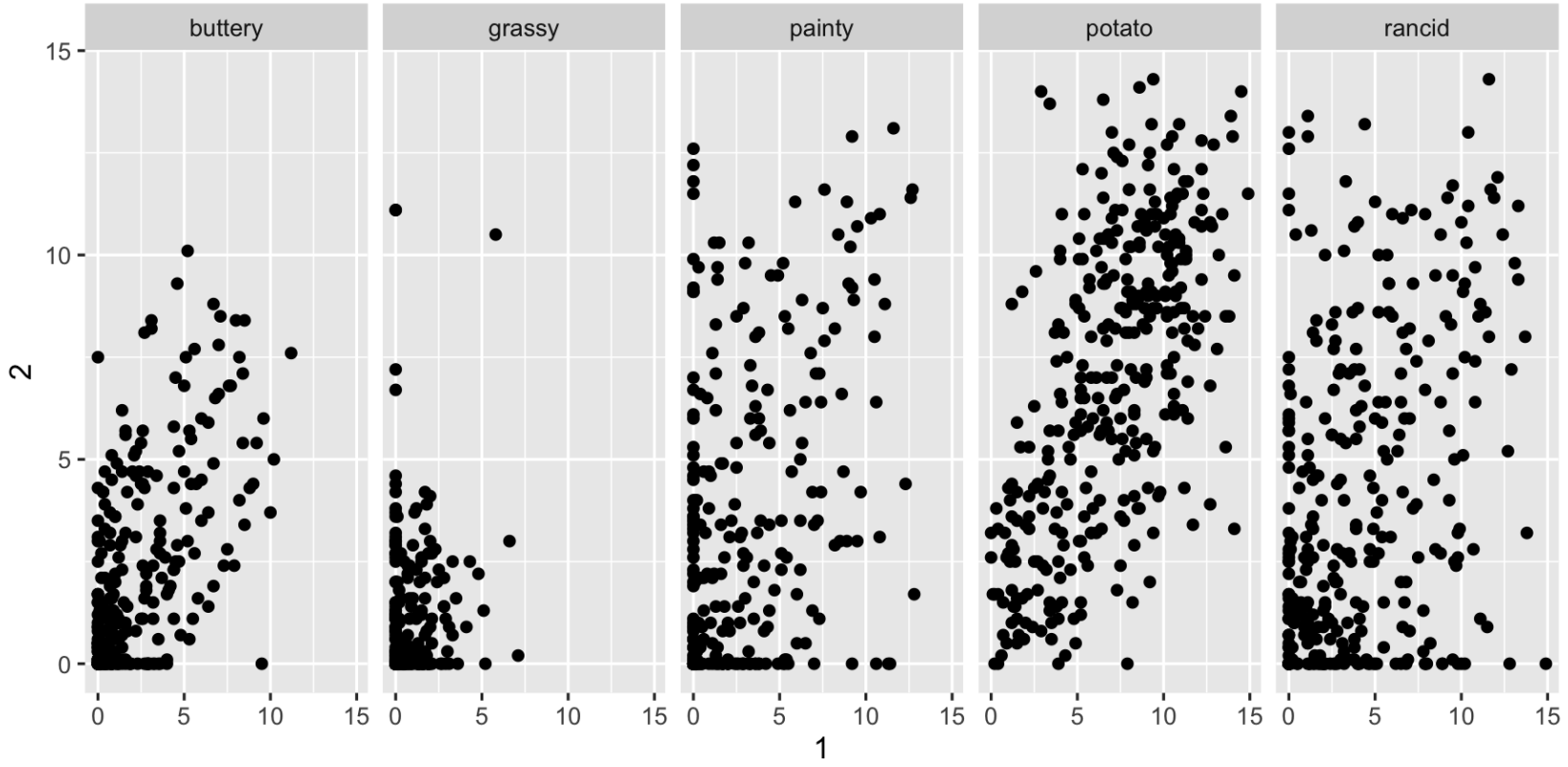
French Fries: Replicates by rating type

```
fries_spread %>%
  group_by(type) %>%
  summarise(r = cor(x = `1`,
                    y = `2`,
                    use = "complete.obs"))
```

```
## # A tibble: 5 x 2
##   type          r
##   <fct>     <dbl>
## 1 buttery 0.650
## 2 grassy  0.239
## 3 painty  0.479
## 4 potato  0.616
```

French Fries: Replicates by rating type

```
ggplot(fries_spread, aes(x=`1`, y=`2`)) +
  geom_point() + facet_wrap(~type, ncol = 5)
```

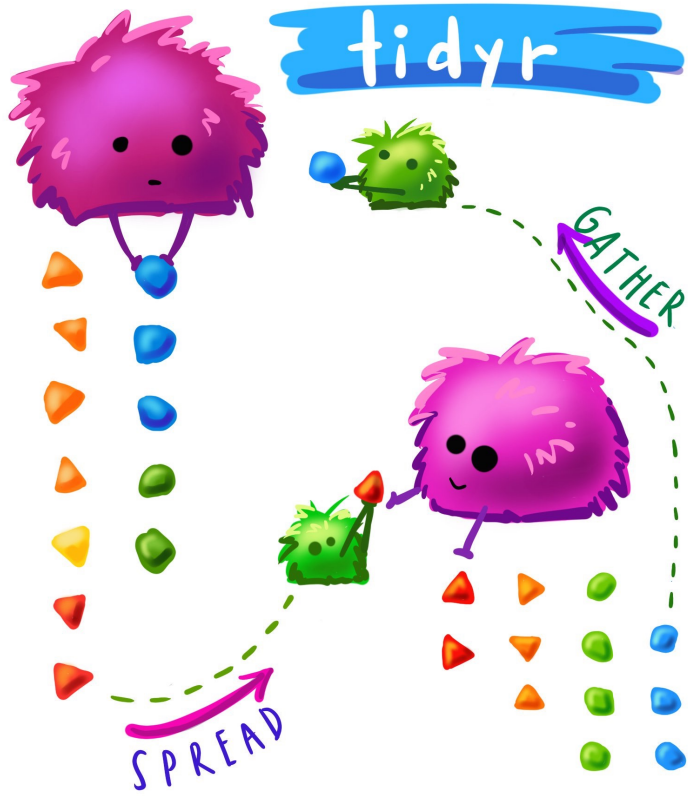


Lab exercise: Exploring data PISA data

Open `pisa.Rmd` on rstudio cloud.

Lab Quiz

Time to take the lab quiz.



Source: A drawing made by Alison Horst

Learning is where you:

1. Receive information accurately